WEST Search History

DATE: Wednesday, September 24, 2003

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L9	L8 not l5	30	L9
L8	l6 and L7	31	L8
L7	l2 and decrement$3	33	L7
L6	l2 and increment$3	63	L6
L5	l2 and L4	7	L5
L4	((711/219)!.CCLS.)	284	L4
L3	l1 and L2	0	L3
L2	((711/110)!.CCLS.)	156	L2
L1	((708/672)!.CCLS.)	96	L1

END OF SEARCH HISTORY

WEST

Generate Collection Print

L9: Entry 9 of 30 File: USPT Dec 26, 2000

US-PAT-NO: 6167488
DOCUMENT-IDENTIFIER: US 6167488 A

TITLE: Stack caching circuit with overflow/underflow unit

DATE-ISSUED: December 26, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Koppala; Sailendra	Mountain View	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Palo Alto	CA			02

APPL-NO: 08/ 828899 [PALM]
DATE FILED: March 31, 1997

PARENT-CASE:
CROSS-REFERENCE TO RELATED APPLICATIONS This application relates to the co-pending application Ser. No. 08/831,279, filed Mar. 31, 1997, entitled "PIPELINED STACK CACHING CIRCUIT", by Koppala, owned by the assignee of this application and incorporated herein by reference. This application also relates to the co-pending application Ser. No. 08/829,105, filed Mar. 31, 1997, entitled "PIPELINED STACK CACHING METHOD", by Koppala, owned by the assignee of this application and incorporated herein by reference. This application also relates to the co-pending application Ser. No. 08/828,769, filed Mar. 31, 1997, entitled "STACK CACHING METHOD WITH OVERFLOW/UNDERFLOW CONTROL", by Koppala, owned by the assignee of this application and incorporated herein by reference.

INT-CL: [07] G06 F 12/12

US-CL-ISSUED: 711/132; 710/53, 710/57, 711/109, 711/110, 711/149, 711/139
US-CL-CURRENT: 711/132; 710/53, 710/57, 711/109, 711/110, 711/139, 711/149

FIELD-OF-SEARCH: 711/139, 711/149, 711/110, 711/109, 711/132, 710/53, 710/57

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3810117	May 1974	Healey	
☐	3878513	April 1975	Werner	
☐	3889243	June 1975	Drimak	
☐	3924245	December 1975	Eaton et al.	
☐	4268903	May 1981	Miki et al.	
☐	4354232	October 1982	Ryan	711/118
☐	4375678	March 1983	Krebs, Jr.	
☐	4524416	June 1985	Stanley et al.	
☐	4530049	July 1985	Zee	711/132
☐	4600986	July 1986	Sheuneman et al.	
☐	4674032	June 1987	Michaelson	
☐	4761733	August 1988	McCrocklin et al.	
☐	4811208	March 1989	Myers et al.	
☐	4951194	August 1990	Bradley et al.	711/132
☐	5043870	August 1991	Ditzel et al.	711/132
☐	5093777	March 1992	Ryan	711/3
☐	5107457	April 1992	Hayes et al.	711/132
☐	5142635	August 1992	Saini	
☐	5157777	October 1992	Lai et al.	
☐	5210874	May 1993	Karger	
☐	5485572	January 1996	Overly	
☐	5535350	July 1996	Maemura	
☐	5603006	February 1997	Satake et al.	
☐	5634027	May 1997	Saito	
☐	5636362	June 1997	Stone et al.	711/129
☐	5687336	November 1997	Shen et al.	
☐	5784553	July 1998	Kolawa et al.	

OTHER PUBLICATIONS

Electronic Engineering, vol. 61, No. 750, Jun. 1989, p. 79, XP000033120, "Up Pops A 32Bit Stack Microprocessor."
Atkinson, R.R., et al., "The Dragon Processor", Second International Conference on Architectural Support for Programming Languages and Operating Systems, No. 1987, Oct. 5, 1987, pp. 65-69, XP000042867.
Stanley, et al., "A Performance Analysis of Automatically Managed Top of Stack Buffers", 14th Annual International Symposium on Computer Architecture, Jun. 2, 1987, pp. 272-281, XP002032257.
Burnley, P: "CPU Architecture for Realtime VME Systems", Microprocessors and Microsystems, London, GB, vol. 12, No. 3; Apr. 1988; pp. 153-158; XP000002633.
Lopriore, L: "Line Fetch/Prefetch in a Stack Cache Memory", Microprocessors and Microystems, vol. 17, No. 9, Nov. 1, 1993, pp. 547-555, XP00413173.

ART-UNIT: 272

PRIMARY-EXAMINER: Peikari; B. James

ATTY-AGENT-FIRM: Gunnison, McKay & Hodgson, L.L.P. Gunnison; Forrest

ABSTRACT:

The present invention provides a stack management unit including a stack cache to accelerate data retrieval from a stack and data storage into the stack. In one embodiment, the stack management unit includes a stack cache, a dribble manager unit, and a stack control unit. The dribble manager unit maintains a cached stack portion, typically a top portion of the stack in the stack cache. The stack cache includes a stack cache memory circuit, one or more read ports, and one or more write ports. The stack management unit also includes an overflow/underflow unit. The overflow/underflow unit detects and resolves overflow conditions and underflow conditions. If an overflow occurs the overflow/underflow unit suspends operation of the stack cache and causes the spill control unit to store the valid data words in the slow memory unit or data cache unit. After the valid data in the stack cache are saved, the overflow/underflow unit equates the cache bottom pointer to the optop pointer. The overflow/underflow unit then resumes normal operation of the stack cache. If an underflow occurs, the overflow/underflow unit suspends operation of the stack cache. In most underflow conditions, data in stack cache 255 are no longer valid and are not saved. Therefore, the overflow/underflow equates the cache bottom pointer to the optop pointer and resumes operation of the stack cache. For underflows caused by context switches, the data in the stack cache must be saved.

29 Claims, 18 Drawing figures

L9: Entry 9 of 30 File: USPT Dec 26, 2000

DOCUMENT-IDENTIFIER: US 6167488 A
TITLE: Stack caching circuit with overflow/underflow unit

Brief Summary Text (13):
The stack cache includes a stack cache memory circuit, one or more read ports, and one or more write ports. The stack cache memory circuit contains a plurality of memory locations, each of which can contain one data word. In one embodiment the stack cache memory circuit is a register file configured with a circular buffer memory architecture. For the circular buffer architecture, the registers can be addressed using modulo addressing. Typically, an optop pointer is used to define and point to the first free memory location in the stack cache memory circuit and a bottom pointer is used to define and point to the bottom memory location in the stack cache memory circuit. As data words are pushed onto or popped off the stack, the optop pointer is incremented or decremented, respectively. Similarly, as data words are spilled or filled between the stack cache memory circuit and the stack, the bottom pointer is incremented or decremented, respectively.

Brief Summary Text (16):
Furthermore, some embodiments of the stack management unit includes an address pipeline to transfer multiple data words by the spill control unit and the fill control unit to improve the throughput of spill and fill operations. The address pipeline contains an incrementor/decrementor circuit, a first address register and a second address register. An address multiplexer drives either the output signal of the incrementor/decrementor or the cache bottom pointer to the first address register. The output terminals of the first address register are coupled to the input terminals of the second address register. A stack cache multiplexer drives either the address in the first address register or the address in the second address register to the stack cache. A memory multiplexer drives either the address in the address multiplexer or in the first address register to the slow memory unit or a data cache unit of the slow memory unit. Furthermore, the address in the second address register can be used to adjust the value in the cache bottom pointer.

Detailed Description Text (22):
Modulo-N increment of X by Y=(X+Y) MOD N,

Detailed Description Text (23):
Modulo-N decrement of X by Y=(X-Y) MOD N.

Detailed Description Text (27):
Pointer OPTOP, pointer OPTOP1, and pointer OPTOP2 are incremented whenever a new data word is written to stack cache 255. Pointer OPTOP, pointer OPTOP1, and pointer OPTOP2 are decremented whenever a stacked data word, i.e., a data word already in stack 180, is popped off stack cache 255. Since some embodiments of stack-based computing system 100 may add or remove multiple data words simultaneously, pointer OPTOP, pointer OPTOP1, and pointer OPTOP2 are implemented, in one embodiment, as programmable registers so that new values can be written into the registers rather than requiring multiple increment or decrement cycles.

Detailed Description Text (28):
If stack cache 255 is organized using sequential addressing, pointer OPTOP1 may also be implemented using a modulo SCS subtractor which modulo-SCS subtracts one from pointer OPTOP. If pointer OPTOP and pointer OPTOP1 are full length memory address pointers, i.e., the pointers address the memory space of stack-based operating system

100 beyond stack cache 255, normal subtraction can be used. For clarity, the various embodiments described herein all use a stack in which addresses are incremented as data is added to the stack. However, the principles of the present invention are easily adaptable to stacks in which addresses are decremented as data is added to the stack.

Detailed Description Text (29):
Since data words are stored in stack cache memory circuit 310 circularly, the bottom of stack cache memory circuit 310 can fluctuate.. Therefore, most embodiments of stack cache management unit 150 include a pointer CACHE.sub.-- BOTTOM to indicate the bottom memory location of stack cache memory circuit 310. Pointer CACHE.sub.-- BOTTOM is typically maintained by dribble manager unit 251. The process to increment or decrement pointer CACHE.sub.-- BOTTOM varies with the specific embodiment of stack cache management unit 150. Pointer CACHE.sub.-- BOTTOM is typically implemented as a programmable up/down counter.

Detailed Description Text (40):
As explained above, stack-based computing system 100 primarily accesses data near the top of the stack. Therefore, stack cache management unit 150 can improve data accesses of stack-based computing system 100 while only caching cached stack portion 182 of stack 180. When stack-based computing system 100 pushes more data words to stack cache management unit 150 than stack cache memory circuit 310 is able to store, the data words near the bottom of stack cache memory circuit 310 are transferred to stack 180 in slow memory unit 190. When stack-based computing system 100 pops data words out of stack cache 255, data words from stack 180 in slow memory unit 190 are copied under the bottom of stack cache memory circuit 310, and pointer CACHE.sub.-- BOTTOM is decremented to point to the new bottom of stack cache memory circuit 310.

Detailed Description Text (50):
Thus, stack cache status circuit 610 can be implemented with a modulo SCS adder/subtractor. The number of used registers USED and the number of free registers FREE can also be generated using a programmable up/down counter. For example, a used register can be incremented whenever a data word is added to stack cache 255 and decremented whenever a data word is removed from stack cache 255. Specifically, if pointer OPTOP is modulo-SCS incremented by some amount, the used register is incremented by the same amount. If pointer OPTOP is modulo-SCS decremented by some amount, the used register is decremented by the same amount. However, if pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented by some amount, the used register is decremented by the same amount. If pointer CACHE.sub.-- BOTTOM is modulo-SCS decremented by some amount, the used register is incremented the same amount. The number of free registers FREE can be generated by subtracting the number of used registers USED from the total number of registers.

Detailed Description Text (54):
FIG. 7A shows another embodiment of dribble manager unit 251, which uses a high-water mark/low-water mark heuristic to determine when a spill condition or a fill condition exists. Spill control unit 394 includes a high water mark register 710 implemented as a programmable up/down counter. A comparator 720 in spill control unit 394 compares the value in high water mark register 710, i.e., the high water mark, with pointer OPTOP. If pointer OPTOP is greater than the high water mark, comparator 720 drives spill signal SPILL to the spill logic level to indicate a spill operation should be performed. Since, the high water mark is relative to pointer CACHE.sub.-- BOTTOM, the high water mark is modulo-SCS incremented and modulo-SCS decremented whenever pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented or modulo-SCS decremented, respectively.

Detailed Description Text (55):
Fill control unit 398 includes a low water mark register 710 implemented as a programmable up/down counter. A comparator 730 in fill control unit 398 compares the value in low water mark register 730, i.e., the low water mark, with pointer OPTOP. If pointer OPTOP is less than the low water mark, comparator 740 drives fill signal FILL to the fill logic level to indicate a fill operation should be performed. Since the low water mark is relative to pointer CACHE.sub.-- BOTTOM, the low water mark register is modulo-SCS incremented and modulo-SCS decremented whenever pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented or modulo-SCS decremented, respectively.

Detailed Description Text (59):
For embodiments of stack cache management unit 150 using saved bits, the saved bit of the register indicated by pointer CACHE.sub.-- BOTTOM is set to the saved logic level. In addition, pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented by one. Other registers as described above may also be modulo-SCS incremented by one. For example, high water mark register 710 (FIG. 7A)and low water mark 730 would be modulo-SCS incremented by one.

Detailed Description Text (60):
Some embodiments of dribble manager unit 251 transfer multiple words for each spill operation, such as the pipelined embodiment of FIG. 8A described below. For these embodiments, pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented by the number words transferred to stack 180 in slow memory unit 190.

Detailed Description Text (61):
In embodiments using a saved bit and valid bit, as shown in FIG. 5, some optimization is possible. Specifically, if the saved bit of the data register pointed to by pointer CACHE.sub.-- BOTTOM is at the saved logic level, the data word in that data register is already stored in stack 180 in slow memory unit 190. Therefore, the data word in that data register does not need to be copied to stack 180 in slow memory unit 190. However, pointer CACHE.sub.-- BOTTOM is still modulo-SCS incremented by one.

Detailed Description Text (63):
Typically, stack cache management unit 150, and more specifically dribble manager unit 251, determines whether the data register preceding the data register pointed by CACHE.sub.-- BOTTOM is free, i.e., either the saved bit is in the saved logic level or the valid bit is in the invalid logic level. If the data register preceding the data register pointed to by pointer CACHE.sub.-- BOTTOM is free, dribble manager unit 251 requests a data word from stack 180 in slow memory unit 190 by sending a request with the value of pointer CACHE.sub.-- BOTTOM modulo-SCS minus one. When the data word is received from data cache unit 160, pointer CACHE.sub.-- BOTTOM is modulo-SCS decremented by one and the received data word is written to the data register pointed to by pointer CACHE.sub.-- BOTTOM through write port 370. Other registers as described above may also be modulo-SCS decremented. The saved bit and valid bit of the register pointed to by pointer CACHE.sub.-- BOTTOM are set to the saved logic level and valid logic level, respectively. Some embodiments of dribble manager unit 251 transfer multiple words for each spill operation. For these embodiments, pointer CACHE.sub.-- BOTTOM is modulo-SCS decremented by the number words transferred to stack 180 in slow memory unit 190.

Detailed Description Text (64):
In embodiments using a saved bit and valid bit, as shown in FIG. 5, some optimization is possible. Specifically, if the saved bit and valid bit of the data register preceding the data register pointed to by pointer CACHE.sub.-- BOTTOM is at the saved logic level and the valid logic level, respectively, then the data word in that data register was never overwritten. Therefore, the data word in that data register does not need to be copied from stack 180 in slow memory unit 190. However, pointer CACHE.sub.-- BOTTOM is still modulo-SCS decremented by one.

Detailed Description Text (65):
IF stack-based computing system 100 operates at a very high frequency, dribble manager unit 251 may not be able to perform the spill and fill functions in one system clock cycle. However, since stack-based computing system 100 reads and writes data from stack cache management unit 150 in one cycle, the latency of a multi-cycle dribble manager unit might be unable to keep pace with stack-based computing system. Furthermore, the latency of a multi-cycle dribble manager unit can cause some cache coherency problems. For example, if a fill condition occurs, pointer CACHE.sub.-- BOTTOM is decremented and the data word corresponding to the new value of pointer CACHE.sub.-- BOTTOM is retrieved from data cache unit 160. If stack-based computing system 100 attempts to read the data word at the new CACHE.sub.-- BOTTOM location after pointer CACHE.sub.-- BOTTOM is decremented but before the data word is retrieved data cache unit 160, stack-based computing system 100 reads incorrect data from stack cache memory circuit 310.

Detailed Description Text (66):
In one embodiment of dribble manager unit 251, both the stack coherency problem and the speed problem of the multi-cycle fill operation are solved by decrementing pointer CACHE.sub.-- BOTTOM only after the data word is retrieved from data cache unit 160. If as in the example above, stack-based computing system 100 reads from what would be the new cache bottom, a stack cache miss occurs so that stack-based computing system 100 must retrieve the data word directly from data cache unit 160. The speed problem is solved by pipelining multiple fill operations whenever a fill operation is required. Specifically, since pointer CACHE.sub.-- BOTTOM is not updated until the data word is retrieved from data cache unit 160, fill control unit 398 detects a fill condition every clock cycle until pointer CACHE.sub.-- BOTTOM is updated to a value which removes the fill condition. Similarly, spill operations are also pipelined to increase the throughput of stack cache 255.

Detailed Description Text (73):
Spill/fill register 850 drives registered spill signal R.sub.-- SPILL and registered fill signal R.sub.-- FILL to INC/DEC circuit 860, which also receives the address in address register 870 as an input signal. INC/DEC circuit 860 functions as an incrementor on spills and a decrementor on fills. Specifically, INC/DEC circuit 860 increments the input value from address register 870 by one if registered spill signal R.sub.-- SPILL is at a spill logic level to indicate a spill condition exists. However, if registered fill signal R.sub.-- FILL is at a fill logic level to indicate a fill condition exists, INC/DEC circuit 860 decrements the input value from address register 870.

Detailed Description Text (83):
At active edge 902, spill/fill register 850 drives registered spill signal R.sub.-- SPILL to the spill logic level (logic high). While registered spill signal R.sub.-- SPILL is at the spill logic level (logic high), INC/DEC circuit 860 increments the address from address register 870. Furthermore, stack cache 255 is given a read signal during every clock cycle registered spill signal R.sub.-- SPILL is at the spill logic level. Similarly, data cache unit 160 is given a write signal during every clock cycle registered spill signal R.sub.-- SPILL is at the spill logic level.

Detailed Description Text (85):
After rising edge 921 of registered spill signal R.sub.-- SPILL, INC/DEC circuit 860 adds one to the output address in address register 870. Thus during clock period 902-P address ID.sub.-- ADDR is 11. Since address register 870 is synchronized with system clock signal S.sub.-- CLK, the contents of address register 870 transition to 11 after active (rising) edge 903. Since the output address in address register 870 serves as an input signal of INC/DEC circuit 860, INC/DEC circuit 860 and address register 870 are incremented every clock cycle that registered spill signal R.sub.-- SPILL is at the spill logic level. After a small propagation delay the contents of address register 870 are sent to stack cache 255 (SC.sub.-- ADDR) and data cache unit 160 (DC.sub.-- ADDR).

Detailed Description Text (89):
FIG. 9B shows a timing diagram for the circuit of FIGS. 8A and 8C for a fill operation. The values in FIG. 9B represent the lower six bits of CACHE.sub.-- BOTTOM, pointer OPTOP, and the various address values. As explained above, during a fill operation address multiplexer 865 outputs address I/D.sub.-- ADDR from INC/DEC circuit 860; memory mux 875 outputs address I/D.sub.-- ADDR from INC/DEC circuit 860; stack cache multiplexer 885 outputs the value from address register 880, and cache bottom multiplexer 805 outputs the address from address register 880. Thus, the simplified circuit of FIG. 8C may help to clarify the timing diagram of FIG. 9B. For the timing diagram of FIG. 9B, pointer CACHE.sub.-- BOTTOM starts with a value of 10, pointer OPTOP reaches a value of 19 at active (rising) edge 951 of system clock signal S.sub.-- CLK, and cache low threshold register contains a value of 10. After pointer OPTOP reaches 19, the number of used registers USED from Modulo SCS subtractor 810 is 9. Therefore, comparator 660 drives fill signal FILL to the fill logic level (logic high in FIG. 9B). At active edge 952, spill/fill register 850 drives registered fill signal R.sub.-- FILL to the fill logic level (logic high). While registered fill signal R.sub.-- FILL is at the fill logic level (logic high), INC/DEC circuit 860 decrements the address from address register 870. Furthermore, data cache unit 160 is given a read signal during every clock cycle registered fill signal R.sub.-- FILL is

at the fill logic level. Similarly, stack cache 255 is given a write signal two clock cycles after the read signal to data cache unit 160. However, if the requested data word is not in data cache unit 160, fill control unit 394 freezes address pipeline 845 and waits until the requested data is retrieved from slow memory unit 190.

Detailed Description Text (90):
After rising edge 971 of registered fill signal R.sub.-- FILL, INC/DEC circuit 860 subtracts one from the output address in address register 870. Thus during clock period 952-P address ID.sub.-- ADDR is 9. Since address register 870 is synchronized with system clock signal S.sub.-- CLK, the contents of address register 870 transition to 9 after active (rising) edge 953. Since the output address in address register 870 serves as the input signal of INC/DEC circuit 860, INC/DEC circuit 860 and address register 870 are decremented every clock cycle that registered fill signal R.sub.-- FILL is at the fill logic level. After a small propagation delay address I/D.sub.-- ADDR is sent to data cache unit 160 (DC.sub.-- ADDR).

Current US Cross Reference Classification (4):
711/110

CLAIMS:

4. The stack cache management unit of claim 3, wherein said optop pointer is incremented when a new data word is pushed to said stack cache memory circuit and wherein said optop pointer is decremented when a popped data word is popped from said stack cache memory circuit.

23. The stack cache management unit of claim 1 wherein said cache bottom pointer is incremented when said spill control unit transfers said first data word from said stack.

24. The stack cache management unit of claim 1 wherein said cache bottom pointer is decremented when said fill control unit transfers said second data word to said stack.

WEST

Generate Collection Print

L9: Entry 11 of 30 File: USPT Oct 10, 2000

US-PAT-NO: 6131144
DOCUMENT-IDENTIFIER: US 6131144 A
**** See image for Certificate of Correction ****

TITLE: Stack caching method with overflow/underflow control using pointers

DATE-ISSUED: October 10, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Koppala; Sailendra	Mountain View	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Palo Alto	CA			02

APPL-NO: 08/ 828769 [PALM]
DATE FILED: April 1, 1997

PARENT-CASE:
CROSS-REFERENCE TO RELATED APPLICATIONS This application relates to the co-pending application Ser. No. 08/831,279, filed Mar. 31, 1997, entitled "PIPELINED STACK CACHING CIRCUIT", by Koppala, owned by the assignee of this application and incorporated herein by reference. This application also relates to the co-pending application Ser. No. 08/829,105, filed Mar. 31, 1997, entitled "PIPELINED STACK CACHING METHOD", by Koppala, owned by the assignee of this application and incorporated herein by reference. This application also relates to the co-pending application Ser. No. 08/828,899, filed Mar. 31, 1997, entitled "STACK CACHING CIRCUIT WITH OVERFLOW/UNDERFLOW UNIT", by Koppala, owned by the assignee of this application and incorporated herein by reference.

INT-CL: [07] G06 F 12/12

US-CL-ISSUED: 711/132; 711/110, 711/109, 710/53, 710/56, 710/57
US-CL-CURRENT: 711/132; 710/53, 710/56, 710/57, 711/109, 711/110

FIELD-OF-SEARCH: 711/109, 711/110, 711/132, 711/139, 711/149, 710/53, 710/56, 710/57

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3810117	May 1974	Healey	
☐	3878513	April 1975	Werner	
☐	3889243	June 1975	Drimak	
☐	3924245	December 1975	Eaton et al.	
☐	4268903	May 1981	Miki et al.	
☐	4354232	October 1982	Ryan	711/118
☐	4375678	March 1983	Krebs, Jr.	
☐	4524416	June 1985	Stanley et al.	
☐	4530049	July 1985	Zee	711/132
☐	4600986	July 1986	Sheuneman et al.	
☐	4674032	June 1987	Michaelson	
☐	4761733	August 1988	McCrocklin et al.	
☐	4811208	March 1989	Myers et al.	
☐	4951194	August 1990	Bradley et al.	711/132
☐	5043870	August 1991	Ditzel et al.	711/132
☐	5093777	March 1992	Ryan	711/3
☐	5107457	April 1992	Hayes et al.	
☐	5142635	August 1992	Saini	
☐	5157777	October 1992	Lai et al.	
☐	5210874	May 1993	Karger	
☐	5485572	January 1996	Overly	
☐	5535350	July 1996	Maemura	
☐	5603006	February 1997	Satake et al.	
☐	5634027	May 1997	Saito	
☐	5636362	June 1997	Stone et al.	711/129
☐	5687336	November 1997	Shen et al.	
☐	5784553	July 1998	Kolawa et al.	

OTHER PUBLICATIONS

Electronic Engineering, vol. 61, No. 750, Jun. 1989, p 79, XP000033120, "Up Pops A 32Bit Stack Microprocessor."
Atkinson, R.R., et al., "The Dragon Processor", Second International Conference on Architectural Support for Programming Languages and Operating Systems, No. 1987, Oct. 1987, pp. 65-69, XP000042867.
Stanley, et al., "A Performance Analysis of Automatically Managed Top of Stack Buffers", 14th Annual International Symposium on Computer Architecture, Jun. 2, 1987, pp. 272-281, XP002032257.
Burnley, P: "CPU Architecture for Realtime VME Systems", Microprocessors and Microsystems, London, GB vol. 12, No. 3; April 1988; pp. 153-158; XP000002633.
Lopriore, L: "Line Fetch/Prefetch in a Stack Cache Memory", Microprocessors and Microsystems, vol. 17, No. 9, Nov. 1, 1993, pp. 547-555, XP00413173.
Microsoft Press Computer Dictionary, 2.sup.nd Ed., p. 279, 1994.

ART-UNIT: 272

PRIMARY-EXAMINER: Peikari; B. James

ATTY-AGENT-FIRM: Gunnison; Forrest

ABSTRACT:

The present invention uses a stack management unit including a stack cache to accelerate data retrieval from a stack and data storage into the stack. In one embodiment, the stack management unit includes a stack cache, a dribble manager unit, and a stack control unit. The dribble manager unit maintains a cached stack portion, typically a top portion of the stack in the stack cache. The stack cache includes a stack cache memory circuit, one or more read ports, and one or more write ports. The stack management unit also includes an overflow/underflow unit. The overflow/underflow unit detects and resolves overflow conditions and underflow conditions. If an overflow occurs the overflow/underflow unit resolves the overflow by suspending operation of the stack cache and spilling a plurality of data words from the stack cache to the stack and equating the bottom pointer to the optop pointer. Typically, the overflow/underflow unit spills all valid data words from the stack cache during an overflow. If an underflow occurs during a context switch the overflow/underflow unit resolves the underflow by spilling a plurality of data word from the stack cache to the stack and equating the bottom pointer to the optop pointer. If an underflow without a context switch the overflow/underflow unit resolves the underflow by equating the bottom pointer to the optop pointer.

24 Claims, 18 Drawing figures

WEST

Generate Collection Print

L9: Entry 11 of 30 File: USPT Oct 10, 2000

DOCUMENT-IDENTIFIER: US 6131144 A
**** See image for Certificate of Correction ****
TITLE: Stack caching method with overflow/underflow control using pointers

Brief Summary Text (13):
The stack cache includes a stack cache memory circuit, one or more read ports, and one or more write ports. The stack cache memory circuit contains a plurality of memory locations, each of which can contain one data word. In one embodiment the stack cache memory circuit is a register file configured with a circular buffer memory architecture. For the circular buffer architecture, the registers can be addressed using modulo addressing. Typically, an optop pointer is used to define and point to the first free memory location in the stack cache memory circuit and a bottom pointer is used to define and point to the bottom memory location in the stack cache memory circuit. As data words are pushed onto or popped off the stack, the optop pointer is incremented or decremented, respectively. Similarly, as data words are spilled or filled between the stack cache memory circuit and the stack, the bottom pointer is incremented or decremented, respectively.

Brief Summary Text (16):
Furthermore, some embodiments of the stack management unit includes an address pipeline to transfer multiple data words by the spill control unit and the fill control unit to improve the throughput of spill and fill operations. The address pipeline contains an incrementor/decrementor circuit, a first address register and a second address register. An address multiplexer drives either the output signal of the incrementor/decrementor or the cache bottom pointer to the first address register. The output terminals of the first address register are coupled to the input terminals of the second address register. A stack cache multiplexer drives either the address in the first address register or the address in the second address register to the stack cache. A memory multiplexer drives either the address in the address multiplexer or in the first address register to the slow memory unit or a data cache unit of the slow memory unit. Furthermore, the address in the second address register can be used to adjust the value in the cache bottom pointer.

Detailed Description Text (25):
Pointer OPTOP, pointer OPTOP1, and pointer OPTOP2 are incremented whenever a new data word is written to stack cache 255. Pointer OPTOP, pointer OPTOP1, and pointer OPTOP2 are decremented whenever a stacked data word, i.e., a data word already in stack 180, is popped off stack cache 255. Since some embodiments of stack-based computing system 100 may add or remove multiple data words simultaneously, pointer OPTOP, pointer OPTOP1, and pointer OPTOP2 are implemented, in one embodiment, as programmable registers so that new values can be written into the registers rather than requiring multiple increment or decrement cycles.

Detailed Description Text (26):
If stack cache 255 is organized using sequential addressing, pointer OPTOP1 may also be implemented using a modulo SCS subtractor which modulo-SCS subtracts one from pointer OPTOP. If pointer OPTOP and pointer OPTOP1 are full length memory address pointers, i.e., the pointers address the memory space of stack-based operating system 100 beyond stack cache 255, normal subtraction can be used. For clarity, the various embodiments described herein all use a stack in which addresses are incremented as data is added to the stack. However, the principles of the present invention are easily adaptable to stacks in which addresses are decremented as data is added to the stack.

Detailed Description Text (27):
Since data words are stored in stack cache memory circuit 310 circularly, the bottom of stack cache memory circuit 310 can fluctuate. Therefore, most embodiments of stack cache management unit 150 include a pointer CACHE.sub.-- BOTTOM to indicate the bottom memory location of stack cache memory circuit 310. Pointer CACHE.sub.-- BOTTOM is typically maintained by dribble manager unit 251. The process to increment or decrement pointer CACHE.sub.-- BOTTOM varies with the specific embodiment of stack cache management unit 150. Pointer CACHE.sub.-- BOTTOM is typically implemented as a programmable up/down counter.

Detailed Description Text (39):
As explained above, stack-based computing system 100 primarily accesses data near the top of the stack. Therefore, stack cache management unit 150 can improve data accesses of stack-based computing system 100 while only caching cached stack portion 182 of stack 180. When stack-based computing system 100 pushes more data words to stack cache management unit 150 than stack cache memory circuit 310 is able to store, the data words near the bottom of stack cache memory circuit 310 are transferred to stack 180 in slow memory unit 190. When stack-based computing system 100 pops data words out of stack cache 255, data words from stack 180 in slow memory unit 190 are copied under the bottom of stack cache memory circuit 310, and pointer CACHE.sub.-- BOTTOM is decremented to point to the new bottom of stack cache memory circuit 310.

Detailed Description Text (48):
Thus, stack cache status circuit 610 can be implemented with a modulo SCS adder/subtractor. The number of used registers USED and the number of free registers FREE can also be generated using a programmable up/down counter. For example, a used register can be incremented whenever a data word is added to stack cache 255 and decremented whenever a data word is removed from stack cache 255. Specifically, if pointer OPTOP is modulo-SCS incremented by some amount, the used register is incremented by the same amount. If pointer OPTOP is modulo-SCS decremented by some amount, the used register is decremented by the same amount. However, if pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented by some amount, the used register is decremented by the same amount. If pointer CACHE.sub.-- BOTTOM is modulo-SCS decremented by some amount, the used register is incremented the same amount. The number of free registers FREE can be generated by subtracting the number of used registers USED from the total number of registers.

Detailed Description Text (52):
FIG. 7A shows another embodiment of dribble manager unit 251, which uses a high-water mark/low-water mark heuristic to determine when a spill condition or a fill condition exists. Spill control unit 394 includes a high water mark register 710 implemented as a programmable up/down counter. A comparator 720 in spill control unit 394 compares the value in high water mark register 710, i.e., the high water mark, with pointer OPTOP. If pointer OPTOP is greater than the high water mark, comparator 720 drives spill signal SPILL to the spill logic level to indicate a spill operation should be performed. Since, the high water mark is relative to pointer CACHE.sub.-- BOTTOM, the high water mark is modulo-SCS incremented and modulo-SCS decremented whenever pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented or modulo-SCS decremented, respectively.

Detailed Description Text (53):
Fill control unit 398 includes a low water mark register 710 implemented as a programmable up/down counter. A comparator 730 in fill control unit 398 compares the value in low water mark register 730, i.e., the low water mark, with pointer OPTOP. If pointer OPTOP is less than the low water mark, comparator 740 drives fill signal FILL to the fill logic level to indicate a fill operation should be performed. Since the low water mark is relative to pointer CACHE.sub.-- BOTTOM, the low water mark register is modulo-SCS incremented and modulo-SCS decremented whenever pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented or modulo-SCS decremented, respectively.

Detailed Description Text (57):
For embodiments of stack cache management unit 150 using saved bits, the saved bit of the register indicated by pointer CACHE.sub.-- BOTTOM is set to the saved logic level. In addition, pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented by one. Other

registers as described above may also be modulo-SCS incremented by one. For example, high water mark register 710 (FIG. 7A)and low water mark 730 would be modulo-SCS incremented by one.

Detailed Description Text (58):
Some embodiments of dribble manager unit 251 transfer multiple words for each spill operation, such as the pipelined embodiment of FIG. 8A described below. For these embodiments, pointer CACHE.sub.-- BOTTOM is modulo-SCS incremented by the number words transferred to stack 180 in slow memory unit 190.

Detailed Description Text (59):
In embodiments using a saved bit and valid bit, as shown in FIG. 5, some optimization is possible. Specifically, if the saved bit of the data register pointed to by pointer CACHE.sub.-- BOTTOM is at the saved logic level, the data word in that data register is already stored in stack 180 in slow memory unit 190. Therefore, the data word in that data register does not need to be copied to stack 180 in slow memory unit 190. However, pointer CACHE.sub.-- BOTTOM is still modulo-SCS incremented by one.

Detailed Description Text (62):
Typically, stack cache management unit 150, and more specifically dribble manager unit 251, determines whether the data register preceding the data register pointed by CACHE.sub.-- BOTTOM is free, i.e., either the saved bit is in the saved logic level or the valid bit is in the invalid logic level. If the data register preceding the data register pointed to by pointer CACHE.sub.-- BOTTOM is free, dribble manager unit 251 requests a data word from stack 180 in slow memory unit 190 by sending a request with the value of pointer CACHE.sub.-- BOTTOM modulo-SCS minus one. When the data word is received from data cache unit 160, pointer CACHE.sub.-- BOTTOM is modulo-SCS decremented by one and the received data word is written to the data register pointed to by pointer CACHE.sub.-- BOTTOM through write port 370. Other registers as described above may also be modulo-SCS decremented. The saved bit and valid bit of the register pointed to by pointer CACHE.sub.-- BOTTOM are set to the saved logic level and valid logic level, respectively. Some embodiments of dribble manager unit 251 transfer multiple words for each spill operation. For these embodiments, pointer CACHE.sub.-- BOTTOM is modulo-SCS decremented by the number words transferred to stack 180 in slow memory unit 190.

Detailed Description Text (63):
In embodiments using a saved bit and valid bit, as shown in FIG. 5, some optimization is possible. Specifically, if the saved bit and valid bit of the data register preceding the data register pointed to by pointer CACHE.sub.-- BOTTOM is at the saved logic level and the valid logic level, respectively, then the data word in that data register was never overwritten. Therefore, the data word in that data register does not need to be copied from stack 180 in slow memory unit 190. However, pointer CACHE.sub.-- BOTTOM is still modulo-SCS decremented by one.

Detailed Description Text (64):
IF stack-based computing system 100 operates at a very high frequency, dribble manager unit 251 may not be able to perform the spill and fill functions in one system clock cycle. However, since stack-based computing system 100 reads and writes data from stack cache management unit 150 in one cycle, the latency of a multi-cycle dribble manager unit might be unable to keep pace with stack-based computing system. Furthermore, the latency of a multi-cycle dribble manager unit can cause some cache coherency problems. For example, if a fill condition occurs, pointer CACHE.sub.-- BOTTOM is decremented and the data word corresponding to the new value of pointer CACHE.sub.-- BOTTOM is retrieved from data cache unit 160. If stack-based computing system 100 attempts to read the data word at the new CACHE.sub.-- BOTTOM location after pointer CACHE.sub.-- BOTTOM is decremented but before the data word is retrieved from data cache unit 160, stack-based computing system 100 reads incorrect data from stack cache memory circuit 310.

Detailed Description Text (65):
In one embodiment of dribble manager unit 251, both the stack coherency problem and the speed problem of the multi-cycle fill operation are solved by decrementing pointer CACHE.sub.-- BOTTOM only after the data word is retrieved from data cache unit 160. If as in the example above, stack-based computing system 100 reads from what would be the

new cache bottom, a stack cache miss occurs so that stack-based computing system 100 must retrieve the data word directly from data cache unit 160. The speed problem is solved by pipelining multiple fill operations whenever a fill operation is required. Specifically, since pointer CACHE.sub.-- BOTTOM is not updated until the data word is retrieved from data cache unit 160, fill control unit 398 detects a fill condition every clock cycle until pointer CACHE.sub.-- BOTTOM is updated to a value which removes the fill condition. Similarly, spill operations are also pipelined to increase the throughput of stack cache 255.

Detailed Description Text (71):
Spill/fill register 850 drives registered spill signal R.sub.-- SPILL and registered fill signal R.sub.-- FILL to INC/DEC circuit 860, which also receives the address in address register 870 as an input signal. INC/DEC circuit 860 functions as an incrementor on spills and a decrementor on fills. Specifically, INC/DEC circuit 860 increments the input value from address register 870 by one if registered spill signal R.sub.-- SPILL is at a spill logic level to indicate a spill condition exists. However, if registered fill signal R.sub.-- FILL is at a fill logic level to indicate a fill condition exists, INC/DEC circuit 860 decrements the input value from address register 870.

Detailed Description Text (81):
At active edge 902, spill/fill register 850 drives registered spill signal R.sub.-- SPILL to the spill logic level (logic high). While registered spill signal R.sub.-- SPILL is at the spill logic level (logic high), INC/DEC circuit 860 increments the address from address register 870. Furthermore, stack cache 255 is given a read signal during every clock cycle registered spill signal R.sub.-- SPILL is at the spill logic level. Similarly, data cache unit 160 is given a write signal during every clock cycle registered spill signal R.sub.-- SPILL is at the spill logic level.

Detailed Description Text (83):
After rising edge 921 of registered spill signal R.sub.-- SPILL, INC/DEC circuit 860 adds one to the output address in address register 870. Thus during clock period 902-P address ID.sub.-- ADDR is 11. Since address register 870 is synchronized with system clock signal S.sub.-- CLK, the contents of address register 870 transition to 11 after active (rising) edge 903. Since the output address in address register 870 serves as an input signal of INC/DEC circuit 860, INC/DEC circuit 860 and address register 870 are incremented every clock cycle that registered spill signal R.sub.-- SPILL is at the spill logic level. After a small propagation delay the contents of address register 870 are sent to stack cache 255 (SC.sub.-- ADDR) and data cache unit 160 (DC.sub.-- ADDR).

Detailed Description Text (88):
FIG. 9B shows a timing diagram for the circuit of FIGS. 8A and 8C for a fill operation. The values in FIG. 9B represent the lower six bits of CACHE.sub.-- BOTTOM, pointer OPTOP, and the various address values. As explained above, during a fill operation address multiplexer 865 outputs address I/D.sub.-- ADDR from INC/DEC circuit 860; memory mux 875 outputs address I/D.sub.-- ADDR from INC/DEC circuit 860; stack cache multiplexer 885 outputs the value from address register 880, and cache bottom multiplexer 805 outputs the address from address register 880. Thus, the simplified circuit of FIG. 8C may help to clarify the timing diagram of FIG. 9B. For the timing diagram of FIG. 9B, pointer CACHE.sub.-- BOTTOM starts with a value of 10, pointer OPTOP reaches a value of 19 at active (rising) edge 951 of system clock signal S.sub.-- CLK, and cache low threshold register contains a value of 10. After pointer OPTOP reaches 19, the number of used registers USED from Modulo SCS subtractor 810 is 9. Therefore, comparator 660 drives fill signal FILL to the fill logic level (logic high in FIG. 9B). At active edge 952, spill/fill register 850 drives registered fill signal R.sub.-- FILL to the fill logic level (logic high). While registered fill signal R.sub.-- FILL is at the fill logic level (logic high), INC/DEC circuit 860 decrements the address from address register 870. Furthermore, data cache unit 160 is given a read signal during every clock cycle registered fill signal R.sub.-- FILL is at the fill logic level. Similarly, stack cache 255 is given a write signal two clock cycles after the read signal to data cache unit 160. However, if the requested data word is not in data cache unit 160, fill control unit 394 freezes address pipeline 845 and waits until the requested data is retrieved from slow memory unit 190.

Detailed Description Text (89):
After rising edge 971 of registered fill signal R.sub.-- FILL, INC/DEC circuit 860 subtracts one from the output address in address register 870. Thus during clock period 952-P address ID.sub.-- ADDR is 9. Since address register 870 is synchronized with system clock signal S.sub.-- CLK, the contents of address register 870 transition to 9 after active (rising) edge 953. Since the output address in address register 870 serves as the input signal of INC/DEC circuit 860, INC/DEC circuit 860 and address register 870 are decremented every clock cycle that registered fill signal R.sub.-- FILL is at the fill logic level. After a small propagation delay address I/D.sub.-- ADDR is sent to data cache unit 160 (DC ADDR).

Detailed Description Paragraph Equation (3):
Modulo-N increment of X by Y=(X+Y)MOD N,

Detailed Description Paragraph Equation (4):
Modulo-N decrement of X by Y=(X-Y)MOD N.

Current US Cross Reference Classification (5):
711/110

CLAIMS:

1. A method for caching a stack in a stack cache having a plurality of memory locations, an optop pointer pointed at a top memory location of said stack cache, and a bottom pointer pointed at a bottom memory location of said stack cache, said method comprising:

writing a new data word for said stack at said top memory location pointed at by said optop pointer;

incrementing said optop pointer;

spilling a first data word stored in said bottom memory pointed at by said bottom pointer from said stack cache to said stack if a spill condition exists;

filling a second data word from said stack to said bottom memory location pointed at by said bottom pointer, or to a memory location adjacent to said bottom memory location of said stack cache if a fill condition exists; and

suspending operation of said stack cache if an overflow condition or an underflow condition exists for said stack cache.

4. The method of claim 3 further comprising:

storing an old optop value when said optop pointer is changed;

spilling one or more data words from said stack cache until said bottom pointer equals said old optop value if said overflow condition exists;

incrementing said bottom pointer; and

equating said bottom pointer to said optop pointer.

13. The method of claim 12 further comprising:

storing an old optop value when said optop pointer is changed;

spilling one or more data words from said stack cache until said bottom pointer equals said old optop value if said underflow condition and said context switch exist;

incrementing said bottom pointer; and

equating said bottom pointer to said optop pointer.

14. The method of claim 1 wherein said spilling a first data word from said stack

cache to said stack comprises:

transferring said first data word from said bottom memory location to said stack; and

incrementing said bottom pointer.

15. The method of claim 1 wherein said filling a second data word from said stack cache to said stack comprises:

decrementing said bottom pointer; and

transferring a second data word from said stack to said bottom memory location.

16. The method of claim 1 further comprising:

reading a data word from said stack cache at a location preceding said top memory location; and

decrementing said optop pointer.

20. The method of claim 19 further comprising:

incrementing said high water mark if said spill condition exists; and

decrementing said high water mark if said fill condition exists.

24. The method of claim 23 further comprising:

incrementing said low water mark if said spill condition exists; and

decrementing said low water mark if said fill condition exists.

WEST

Generate Collection | Print

L9: Entry 17 of 30 File: USPT Jul 13, 1999

US-PAT-NO: 5924114
DOCUMENT-IDENTIFIER: US 5924114 A

TITLE: Circular buffer with two different step sizes

DATE-ISSUED: July 13, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Maruyama; Toshiyuki	Tokyo			JP
Matsuo; Masahito	Tokyo			JP

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Mitsubishi Denki Kabushiki Kaisha	Tokyo			JP	03

APPL-NO: 08/ 890618 [PALM]
DATE FILED: July 9, 1997

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
JP	9-034960	February 19, 1997

INT-CL: [06] G06 F 12/00

US-CL-ISSUED: 711/110; 711/213
US-CL-CURRENT: 711/110; 711/213

FIELD-OF-SEARCH: 711/213, 711/217, 711/110

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4908748	March 1990	Pathak et al.	711/220
☐	5463749	October 1995	Wertheizer et al.	711/110
☐	5623621	April 1997	Garde	711/110 X

OTHER PUBLICATIONS

U.S. Patent application Ser. No. 08/699,944 filed Aug. 20, 1996.
TMS320C54X User's Guide, Texas Instruments, 1995, pp. 7-22 to 7-25.
DSP1610 Architecture, Instruction Set, and Development Tools, Version 1.0, AT&T, 1992, pp. 7-29 to 7-33.

ART-UNIT: 272

PRIMARY-EXAMINER: Robertson; David L.

ATTY-AGENT-FIRM: Burns, Doane, Swecker & Mathis, L.L.P

ABSTRACT:

A control unit (112) makes different judgments on the end address, depending on whether 1-word access or 2-word access, based on a post-update signal (507) and a 2-word access signal (508) which are internally generated and a coincidence signal (511) on the high-order 14 bits and another coincidence signal (512) on the bit 14 which are outputted from a comparator (158), and outputs a judgment result to a selector (155) as a selection signal (510). The selector (155) selects one of an output from an ALU (153) and an output from a latch (159) (the MOD.sub.-- S register 156) based on the selection signal (510). Having this structure, a data processor which enables access with modulo addressing in two different data-units can be provided.

19 Claims, 40 Drawing figures

L9: Entry 17 of 30 File: USPT Jul 13, 1999

DOCUMENT-IDENTIFIER: US 5924114 A
TITLE: Circular buffer with two different step sizes

Brief Summary Text (8):
Another type of DSP judges whether to modify an address based on the address before updating of the pointer. The DSP holds a beginning address (rb) and an ending address (re) to define the range of the circular buffer region. When the postincremented address coincides with the ending address (re), the beginning address (rb) is written back as an updated address. This type of DSP, however, has a problem that the modulo addressing can work only when a 1-increment is made.

Brief Summary Text (9):
In an operation of the DSP, since the region to be accessed with modulo addressing is designated by word address, only 1-word access is allowed with auto-increment or auto-decrement and only one setting for modulo addressing is made. Further, if 2-word access is made to the region with modulo addressing, no other access than 2-word access can be made.

Brief Summary Text (10):
Furthermore, there arises an inconvenience of setting start address and end address since the start address and the end address are reversed between increment and decrement.

Brief Summary Text (13):
According to a second aspect of the present invention, in the data processor of the first aspect, the first memory-access instruction includes a first increment operation for incrementing an address to accomplish the updating of address, the second memory-access instruction includes a second increment operation for incrementing an address to accomplish the updating of address, and the address-update direction is a direction that increments an address. Further, the start-address information supply means includes start-address information holding means for holding the start-address information, and the end-address information supply means includes end-address information holding means for holding the end-address information.

Brief Summary Text (16):
According to a fifth aspect of the present invention, in the data processor of the first aspect, the first memory-access instruction includes a first decrement operation for decrementing an address to accomplish the updating of address, the second memory-access instruction includes a second decrement operation for decrementing an address to accomplish the updating of address, and the address-update direction is a direction that decrements an address. Further, the start-address information supply means includes start-address information holding means for holding the start-address information, and the end-address information supply means includes end-address information holding means for holding the end-address information.

Brief Summary Text (19):
According to an eighth aspect of the present invention, the data processor of the first aspect further comprises: low-limit address holding means for holding a low limit address of the circular buffer region; and high-limit address holding means for holding a high limit address of the circular buffer region. In the data processor of the eighth aspect, the first memory-access instruction includes a first increment operation for incrementing an address to accomplish the updating of address, the second memory-access instruction includes a second increment operation for

incrementing an address to accomplish the updating of address, and the address-update direction is a direction that increments an address when the first and second memory-access instructions perform the first and second increment operations, respectively. Further in the data processor of the eighth aspect, the first memory-access instruction includes a first decrement operation for decrementing an address to accomplish the updating of address, the second memory-access instruction includes a second decrement operation for decrementing an address to accomplish the updating of address, and the address-update direction is a direction that decrements an address when the first and second memory-access instructions perform the first and second decrement operations, respectively. Still further in the data processor of the eighth aspect, the start-address information supply means includes first selection means receiving executable-instruction information indicating a content of the first or second memory-access instruction to be executed and the low and high limit addresses, for supplying the start-address information designating the low limit address when the executable-instruction information indicates the first or second increment operation and supplying the start-address information designating the high limit address when the executable-instruction information indicates the first or second decrement operation, and the end-address information supply means includes second selection means receiving the executable-instruction information and the low and high limit addresses, for supplying the end-address information designating the high limit address when the executable-instruction information indicates the first or second increment operation and supplying the end-address information designating the low limit address when the executable-instruction information indicates the first or second decrement operation.

Brief Summary Text (20):
According to a ninth aspect of the present invention, in the data processor of the eighth aspect, the low limit address and the high limit address each include an address which can specify n-bit data in the circular buffer region, the n-bit access address includes the address designated by the start-address information, and the 2n-bit access address is the address designated by the start-address information when the second memory-access instruction performs the second increment operation and the 2n-bit access address includes an address field to specify 2n-bit data of the address designated by the start-address information and fixed data to enable access in units of 2n-bit data when the second memory-access instruction performs the second decrement operation.

Brief Summary Text (21):
According to a tenth aspect of the present invention, in the data processor of the eighth aspect, the low limit address and the high limit address each include an address which can specify 2n-bit data in the circular buffer region, the first condition used when the access-address selection means selects the next access address is a condition that the access address coincides with an address specifying the second one of two n-bit data constituting 2n-bit data in the circular buffer region specified by the address designated by the end-address information when the first memory-access performs the first increment operation, the n-bit access address is the address designated by the start-address information when the first memory-access instruction performs the first increment operation and the n-bit access address is an address specifying the second one of two n-bit data constituting 2n-bit data in the circular buffer region specified by the address designated by the start-address information when the first memory-access instruction performs the first decrement operation, and the 2n-bit access address includes the address designated by the start-address information.

Brief Summary Text (24):
The present invention is directed to a data processor having a memory with a circular buffer region to be accessed in units of n-bit data and being capable of executing at least a memory-access instruction which performs access to the memory in units of n-bit data with updating of address to specify n-bit data to be next accessed. According to a thirteenth aspect of the present invention, in the data processor, the memory-access instruction includes an increment operation for incrementing an address to accomplish the updating of address or a decrement operation for decrementing an address to accomplish the updating of address, and an address-update direction is a direction that increments an address when the memory-access instruction performs the increment operation and is a direction that decrements an address when the

memory-access instruction performs the decrement operation. The data processor of the thirteenth aspect comprises: low-limit address holding means for holding a low limit address of the circular buffer region; and high-limit address holding means for holding a high limit address of the circular buffer region. The low limit address and the high limit address each include an address which can specify n-bit data in the circular buffer region. The data processor of the thirteenth aspect further comprises: first selection means receiving executable-instruction information indicating a content of the memory-access instruction to be executed and the low and high limit addresses, for supplying start-address information designating the low limit address when the executable-instruction information indicates the increment operation and supplying the start-address information designating the high limit address when the executable-instruction information indicates the decrement operation; and second selection means receiving executable-instruction information and the low and high limit addresses, for supplying end-address information designating the high limit address when the executable-instruction information indicates the increment operation and supplying the end-address information designating the low limit address when the executable-instruction information indicates the decrement operation. In the data processor of the thirteenth aspect, a direction from an address designated by the start-address information to an address designated by the end-address information is defined the address-update direction. The data processor of the thirteenth aspect further comprises: comparison means for comparing an access address specifying data to be accessed in the circular buffer region with the address designated by the end-address information to output comparison result information; address calculation means for calculating the next address of the access address in the address-update direction to output a calculated address during execution of the memory-access instruction; and access-address selection means for selecting the address designated by the start-address information or the calculated address on the basis of the comparison result information as a next access address to specify data to be next accessed. The access-address selection means selects the address designated by the start-address information as the next access address if it judges that a first condition that the address designated by the end-address information and the access address substantially coincide in address field to specify n-bit data is true on the basis of the comparison result information, and selects the calculated address as the next access address if it judges that the first condition is not true.

Brief Summary Text (29):
In the data processor of the second aspect, the first and second memory-access instructions have the first and second increment operations, respectively, for incrementing an address to accomplish updating of address, and the address-update direction is a direction that increments an address.

Brief Summary Text (35):
In the data processor of the fifth aspect, the first and second memory-access instructions have the first and second decrement operations, respectively, for decrementing an address to accomplish updating of address, and the address-update direction is a direction that decrements an address.

Brief Summary Text (42):
In the data processor of the eighth aspect, the start-address information supply means includes first selection means for supplying the start-address information designating the low limit address when the executable-instruction information indicates the first or second increment operation and supplying the start-address information designating the high limit address when the executable-instruction information indicates the first or second decrement operation, and the end-address information supply means includes second selection means for supplying the end-address information designating the high limit address when the executable-instruction information indicates the first or second increment operation and supplying the end-address information designating the low limit address when the executable-instruction information indicates the first or second decrement operation.

Brief Summary Text (43):
Therefore, in the data processor of the eighth aspect, the first and second selection means can automatically supply correct start-address information and end-address information, whether the first and second memory-access instructions perform the first and second increment operations or the first and second decrement operations, by

storing the low limit address and the high limit address of the circular buffer region into the low-limit address holding means and the high-limit address holding means.

Brief Summary Text (44):
As a result, the data processor of the eighth aspect eliminates the need for changing addresses to be stored in the low-limit address holding means and the high-limit address holding means depending on whether the first and second memory-access instructions perform the first and second increment operations or the first and second decrement operations, and accordingly it can reduce the code size of program and the number of operation cycles, to thereby achieve high-speed operation.

Brief Summary Text (47):
Since the 2n-bit access address is the address designated by the start-address information when the second memory-access instruction performs the second increment operation and the 2n-bit access address includes an address field to specify 2n-bit data of the address designated by the start-address information and fixed data to enable access in units of 2n-bit data when the second memory-access instruction performs the second decrement operation, the access-address selection means can correctly select the 2n-bit access address as the next access address when the second condition is true.

Brief Summary Text (49):
Since the first condition used when the access-address selection means selects the next access address includes a condition that the access address coincides with an address specifying the second one of two n-bit data constituting 2n-bit data in the circular buffer region specified by the address designated by the end-address information when the first memory-access instruction performs the first increment operation, the access-address selection means can make a correct judgment on the end address of the first memory-access instruction.

Brief Summary Text (50):
Further, the n-bit access address is the address designated by the start-address information when the first memory-access instruction performs the first increment operation and the n-bit access address is an address specifying the second one of two n-bit data constituting 2n-bit data in the circular buffer region specified by the address designated by the start-address information when the first memory-access instruction performs the first decrement operation. Therefore, the access-address selection means can correctly select the n-bit access address as the next access address when the first condition is true.

Brief Summary Text (53):
The data processor of the thirteenth aspect comprises first selection means for supplying start-address information designating the low limit address when the executable-instruction information indicates the increment operation and supplying the start-address information designating the high limit address when the executable-instruction information indicates the decrement operation, and second selection means for supplying end-address information designating the high limit address when the executable-instruction information indicates the increment operation and supplying the end-address information designating the low limit address when the executable-instruction information indicates the decrement operation.

Brief Summary Text (54):
Therefore, in the data processor of the thirteenth aspect, the first and second selection means can automatically supply correct start-address information and end-address information, whether the first and second memory-access instructions perform the first and second increment operations or the first and second decrement operations, by storing the low limit address and the high limit address of the circular buffer region into the low-limit address holding means and the high-limit address holding means.

Brief Summary Text (55):
As a result, the data processor of the thirteenth aspect eliminates the need for changing addresses to be stored in the low-limit address holding means and the high-limit address holding means depending on whether the first and second memory-access instructions perform the first and second increment operations or the

first and second decrement operations, and accordingly it can reduce the code size of program and the number of operation cycles, to thereby achieve efficient modulo addressing over the circular buffer region.

Brief Summary Text (58):
An object of the present invention is to provide a data processor which enables accesses in two different data-units with modulo addressing or a data processor which eliminates the inconvenience of setting start address and end address, which is needed in switching between increment and decrement.

Detailed Description Text (11):
The registers CR10 and CR11 are control registers for modulo addressing. The register CR10 holds a modulo start address (MOD.sub.-- S) and the register CR11 holds a modulo end address (MOD.sub.-- E). The registers CR10 and CR11 hold the first and the last data word addresses (16 bits), respectively. In the modulo addressing with increment, the lower address is designated as the modulo start address MOD.sub.-- S and the higher address is designated as the modulo end address MOD.sub.-- E. When a register value to be incremented coincides with the modulo end address MOD.sub.-- E, the address value of the modulo start address MOD.sub.-- S is written back to the register as an increment result.

Detailed Description Text (31):
The ALU 153 mainly performs transfer, comparison, arithmetic and logic operation, calculation/transfer of operand addresses, increment/decrement of operand address values, calculation/transfer of jump target addresses and the like. Results of operation and address modification are transferred through a selector 155 over the D1 bus 311 and written back to the register designated by the instruction in the register file 115. For example, to perform increment (decrement), the ALU 153 adds the base address held in the AA latch 151 and the increment (decrement) address value held in the AB latch 152. The AB latch 152 holds "4" for 2-word access and holds "2" for 1-word access.

Detailed Description Text (37):
FIG. 10 is a block diagram showing a detailed structure of a program counter (PC) unit 118. An instruction address (IA) register 181 holds the address of the next instruction to be fetched and outputs the address to the instruction fetch unit 102. When a subsequent instruction is to be fetched, the address value transferred from the IA register 181 through a latch 182 is incremented by 1 in an incrementor 183 and then written back into the IA register 181. If a jump or repeat instruction changes the sequence, the IA register 181 receives the jump target address or the repeat block start address transferred over the JA bus 323.

Detailed Description Text (38):
An RPT.sub.-- S register 184, an RPT.sub.-- E register 186 and an RPT.sub.-- C register 188 are repeat control registers and correspond to the control registers CR8, CR9 and CR7 of FIG. 1, respectively. The RPT.sub.-- E register 186 holds the address of the last instruction in the block to be repeated. The last address is calculated by the first operation unit 116 during execution of the repeat instruction and given to the RPT.sub.-- E register 186 over the JA bus 323. A comparator 187 compares the value of an end address in the block to be repeated which is held in the RPT.sub.-- E register 186 with the value of a fetch address which is held in the IA register 181. If the value of a repeat count which is held in the RPT.sub.-- C register 188 is not "1" during execution of the repeat instruction and the two addresses coincide with each other, the value of a start address in the block to be repeated which is held in the RPT.sub.-- S register 184 is transferred to the IA register 181 through a latch 185 over the JA bus 323. Every time the last instruction of the block to be repeated is executed, the value held in the RPT.sub.-- C register 188 is decremented by 1 in a decrementor 190 through a latch 189. If the decremented value is "0", the RP bit 43 of the PSW is cleared and the execution of the repeat instruction is terminated. The RPT.sub.-- S register 184, the RPT.sub.-- E register 186 and the RPT.sub.-- C register 188 each have an input port connected to the D1 bus 311 and an output port connected to the S3 bus 303. By using these buses, initialization caused by execution of the repeat instruction, and saving and returning operations are performed as required.

Detailed Description Text (39):

An execution-stage PC (EPC) 194 holds the PC value of the instruction being executed, and a next-instruction PC (NPC) 191 holds the PC value of the next instruction. If a jump occurs during execution, the NPC 191 receives the jump target address value on the JA bus 323. If a branch occurs during repeat operation, the NPC 191 receives the first address in the block to be repeated from the latch 185, and otherwise the value of the NPC 191 is transferred through a latch 192 to an incrementor 193 which increments it and then written back into the NPC 191. When a subroutine jump instruction is executed, the value held in the latch 192 is given to the D1 bus 311 as a return address and then written back into the register R13 designated as a link register in the register file 115. When the next instruction comes into execution, the value held in the latch 192 is transferred to the EPC 194. To make a reference to the PC value of the instruction being executed, the value held in the EPC 194 is transferred to the first operation unit 116 over the S3 bus 303. A BPC 196 corresponds to the control register CR3 in the register set of FIG. 1. When an exception or interruption is found, the value held in the EPC 194 is transferred to the BPC 196 through a latch 195. The BPC 196 has an input port connected to the D1 bus 311 and an output port connected to the S3 bus 303, and performs saving and returning as required.

Detailed Description Text (51):
In the IF stage 401, mainly, a fetch of instructions, management of the instruction queue 111 and repeat control are performed. The IF stage 401 controls the operations of the instruction fetch unit 102, the integrated instruction memory 103, the external bus interface unit 106, the instruction queue 111, the IA register 181, the latch 182, the incrementor 183 and the comparator 187 in the PC unit 118, and parts of the control unit 112 to achieve an IF stage control, an instruction fetch control and a control of the PC unit 118. The IF stage 401 is initialized by a jump at the E stage 403.

Detailed Description Text (52):
A fetch address is held in the IA register 181. If a jump occurs in the E stage 403, the IA register 181 receives the jump target address over the JA bus 323 and perform initialization. To sequentially fetch the instruction data, the incrementor 183 increments the address. During execution of a repeat instruction, if the comparator 187 detects coincidence between the value held in the IA register 181 and the value held in the RPT.sub.-- E register 186 and the value held in the RPT.sub.-- C register 188 is not "1", the sequence is controlled to change over. In this case, the value held in the RPT.sub.-- S register 184 is transferred to the IA register 181 through the latch 185 over the JA bus 323.

Detailed Description Text (142):
Though the data processor of the first preferred embodiment discussed above can perform its function in both increment and decrement, the data processor may be structured so as to perform its function in either increment or decrement.

Detailed Description Text (183):
Moreover, the second preferred embodiment ensures proper postincrement/postdecrement operation by setting the high limit word address and the low limit word address of the circular buffer region to the MOD.sub.-- U register 650 and the MOD.sub.-- L register 651 respectively, regardless of whether postincrement or postdecrement, unlike the first preferred embodiment, and therefore it eliminates the need for resetting the control register in switching between the increment and decrement. That enables further reduction in code size of the program and the number of operation cycles.

Detailed Description Text (198):
When modulo addressing is performed with increment, the lower address is set to the MOD.sub.-- S register 851 and the higher address is set to the MOD.sub.-- E register 850, and when modulo addressing is performed with decrement, the higher address is set to the MOD.sub.-- S register 851 and the lower address is set to the MOD.sub.-- E register 850. In the data processor of the third preferred embodiment, the boundaries of the circular buffer must be aligned in 2 words.

Detailed Description Text (231):
Though the data processor of the third preferred embodiment discussed above can perform its function in both increment and decrement, the data processor may be

structured so as to perform its function in either increment or decrement.

Current US Original Classification (1):
711/110

CLAIMS:

2. The data processor of claim 1, wherein

said first memory-access instruction includes a first increment operation for incrementing an address to accomplish said updating of address, said second memory-access instruction includes a second increment operation for incrementing an address to accomplish said updating of address, and said address-update direction is a direction that increments an address,

said start-address information supply means includes start-address information holding means for holding said start-address information, and

said end-address information supply means includes end-address information holding means for holding said end-address information.

3. The data processor of claim 2, wherein

said first memory-access instruction operates said first increment operation after accessing said memory, and said second memory-access instruction operates said second increment operation after accessing said memory.

6. The data processor of claim 1, wherein

said first memory-access instruction includes a first decrement operation for decrementing an address to accomplish said updating of address, said second memory-access instruction includes a second decrement operation for decrementing an address to accomplish said updating of address, and said address-update direction is a direction that decrements an address,

said start-address information supply means includes start-address information holding means for holding said start-address information, and

said end-address information supply means includes end-address information holding means for holding said end-address information.

7. The data processor of claim 6, wherein

said first memory-access instruction operates said first decrement operation after accessing said memory, and said second memory-access instruction operates said second decrement operation after accessing said memory.

10. The data processor of claim 1, further comprising:

low-limit address holding means for holding a low limit address of said circular buffer region; and

high-limit address holding means for holding a high limit address of said circular buffer region,

wherein said first memory-access instruction includes a first increment operation for incrementing an address to accomplish said updating of address, said second memory-access instruction includes a second increment operation for incrementing an address to accomplish said updating of address, and said address-update direction is a direction that increments an address when said first and second memory-access instructions perform said first and second increment operations, respectively,

wherein said first memory-access instruction includes a first decrement operation for decrementing an address to accomplish said updating of address, said second memory-access instruction includes a second decrement operation for decrementing an

address to accomplish said updating of address, and said address-update direction is a direction that decrements an address when said first and second memory-access instructions perform said first and second decrement operations, respectively,

and wherein said start-address information supply means includes first selection means receiving executable-instruction information indicating a content of said first or second memory-access instruction to be executed and said low and high limit addresses, for supplying said start-address information designating said low limit address when said executable-instruction information indicates said first or second increment operation and supplying said start-address information designating said high limit address when said executable-instruction information indicates said first or second decrement operation, and

said end-address information supply means includes second selection means receiving said executable-instruction information and said low and high limit addresses, for supplying said end-address information designating said high limit address when said executable-instruction information indicates said first or second increment operation and supplying said end-address information designating said low limit address when said executable-instruction information indicates said first or second decrement operation.

11. The data processor of claim 10, wherein

said low limit address and said high limit address each include an address which can specify n-bit data in said circular buffer region,

said n-bit access address includes said address designated by said start-address information, and

said 2n-bit access address is said address designated by said start-address information when said second memory-access instruction performs said second increment operation and said 2n-bit access address includes an address field to specify 2n-bit data of said address designated by said start-address information and fixed data to enable access in units of 2n-bit data when said second memory-access instruction performs said second decrement operation.

12. The data processor of claim 10, wherein

said low limit address and said high limit address each include an address which can specify 2n-bit data in said circular buffer region,

said first condition used when said access-address selection means selects said next access address is a condition that said access address coincides with an address specifying the second one of two n-bit data constituting 2n-bit data in said circular buffer region specified by said address designated by said end-address information when said first memory-access performs said first increment operation,

said n-bit access address is said address designated by said start-address information when said first memory-access instruction performs said first increment operation and said n-bit access address is an address specifying the second one of two n-bit data constituting 2n-bit data in said circular buffer region specified by said address designated by said start-address information when said first memory-access instruction performs said first decrement operation, and

said 2n-bit access address includes said address designated by said start-address information.

15. A data processor having a memory with a circular buffer region to be accessed in units of n-bit data and being capable of executing at least a memory-access instruction which performs access to said memory in units of n-bit data with updating of address to specify n-bit data to be next accessed,

said memory-access instruction including an increment operation for incrementing an address to accomplish said updating of address or a decrement operation for decrementing an address to accomplish said updating of address,

wherein an address-update direction is a direction that increments an address when said memory-access instruction performs said increment operation and is a direction that decrements an address when said memory-access instruction performs said decrement operation,

said data processor comprising:

low-limit address holding means for holding a low limit address of said circular buffer region; and

high-limit address holding means for holding a high limit address of said circular buffer region, said low limit address and said high limit address each including an address which can specify n-bit data in said circular buffer region,

said data processor further comprising:

first selection means receiving executable-instruction information indicating a content of said memory-access instruction to be executed and said low and high limit addresses, for supplying start-address information designating said low limit address when said executable-instruction information indicates said increment operation and supplying said start-address information designating said high limit address when said executable-instruction information indicates said decrement operation; and

second selection means receiving executable-instruction information and said low and high limit addresses, for supplying end-address information designating said high limit address when said executable-instruction information indicates said increment operation and supplying said end-address information designating said low limit address when said executable-instruction information indicates said decrement operation,

wherein a direction from an address designated by said start-address information to an address designated by said end-address information is defined said address-update direction,

said data processor further comprising:

comparison means for comparing an access address specifying data to be accessed in said circular buffer region with said address designated by said end-address information to output comparison result information;

address calculation means for calculating the next address of said access address in said address-update direction to output a calculated address during execution of said memory-access instruction; and

access-address selection means for selecting said address designated by said start-address information or said calculated address on the basis of said comparison result information as a next access address to specify data to be next accessed,

wherein said access-address selection means selects said address designated by said start-address information as said next access address if judges that a first condition is true on the basis of said comparison result information, said first condition being that said address designated by said end-address information and said access address substantially coincide in address field to specify n-bit data, and selects said calculated address as said next access address if judges that said first condition is not true.

16. The data processor of claim 15, wherein

said memory-access instruction performs said increment operation after accessing said memory.

17. The data processor of claim 15, wherein

said memory-access instruction performs said decrement operation after accessing said

memory.

WEST

Generate Collection | Print

L9: Entry 18 of 30 File: USPT Feb 23, 1999

US-PAT-NO: 5875463
DOCUMENT-IDENTIFIER: US 5875463 A

TITLE: Video processor with addressing mode control

DATE-ISSUED: February 23, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Crump; Dwayne T.	Apex	NC		
Pancoast; Steve T.	Raleigh	NC		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 08/ 907034 [PALM]
DATE FILED: August 6, 1997

PARENT-CASE:
This application is a continuation of application Ser. No. 08/472,208, filed Jun. 7, 1995 now abandoned.

INT-CL: [06] G06 F 12/08

US-CL-ISSUED: 711/123; 711/1, 711/110, 395/800.01, 345/519
US-CL-CURRENT: 711/123; 345/519, 711/1, 711/110

FIELD-OF-SEARCH: 711/1, 711/123, 711/110, 345/28, 345/193, 345/515, 345/516, 345/502, 345/501, 345/519, 395/800.1

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4606066	August 1986	Hata et al.	382/304
☐	5058065	October 1991	D'Luna	365/189.02
☐	5239654	August 1993	Ing-Simmons et al.	365/800.2
☐	5390304	February 1995	Leach et al.	395/588
☐	5394519	February 1995	Bodin	345/431
☐	5471592	November 1995	Goue et al.	395/200.43
☐	5479166	December 1995	Read et al.	341/65

OTHER PUBLICATIONS

K. Ishida et al; A 10-GHz 8-b Multiplexer/Demultiplexer Chip Set for the SONET STS-192 System; IEE Journal of Solid-State Circuits 26 No.12, pp. 1936-1942 (Dec. 1991).
IBM Technical Disclosure Bulletin, vol. 36 No. 3, pp. 397-400 J. Pershing et al; Prefetching for a Chain of Control Blocks; Mar. 1993.
IBM Technical Disclosure Bulletin, vol. 36 No. 3, pp. 507-508; J. Pomerene and R. Rechtschaffen; Prefetching for the Iterations of an Indexed Loop; Mar. 1993.
IBM Technical Disclosure Bulletin, vol. 36 No. 12, pp. 119-120; K. Ekanadham et al; One-Chip Processors and an Unconditional Branch; Dec. 1993.

ART-UNIT: 271

PRIMARY-EXAMINER: Chan; Eddie P.

ASSISTANT-EXAMINER: Kim; Hong C.

ATTY-AGENT-FIRM: McConnell; Daniel E. Magistrale; Anthony N.

ABSTRACT:

Advantage is taken of Very Large Scale Integrated (VLSI) circuit design and manufacture to provide, in a digital data handling system handling display signal streams, a video processor which is capable of high performance due to vector processing and special addressing modes. The video processor has, on a single VLSI device, a plurality of processors which cooperate for generating video signal streams and which employ distinctive addressing modes for memory elements of the device. Each of the plurality of processors has associated instruction and data caches, which are joined together by a wide data bus formed on the same substrate as the processors, and further has registers for controlling access, and the modes of access, to data held in memory.

38 Claims, 12 Drawing figures

L9: Entry 18 of 30 File: USPT Feb 23, 1999

DOCUMENT-IDENTIFIER: US 5875463 A
TITLE: Video processor with addressing mode control

Detailed Description Text (106):
The instruction decoder (indicated at 42a in FIG. 8) makes use of simple decoders and demultiplexers to control the ALU and Load/Store unit operations. On each clock edge, the result of the ALU and Load/Store unit is latched into the appropriate register and a new instruction is latched into the decode buffer and the IP is auto incremented. The decode buffer always holds the current instruction while the IP has already been incremented (on the clock edge) and is pointing to the next instruction. This causes the next instruction to be latched into the decode buffer even if the current instruction is a jump instruction and modifies the IP. I.E. the next instruction after a jump instruction is always executed. However, the I Cache gets the next instruction address early and has time to present the requested data. Greater depth of instruction look ahead was not needed due to the prefetch algorithm used in the I Cache.

Detailed Description Text (139):
The index register is automatically incremented or decremented by the word size on every access to memory via the Load/Store unit. The enable/disable bit in the ICR is used to control this function. The increment/decrement bit is used to control the direction of the index pointer. The post/pre bit is used to control when the automatic increment/decrement takes place (either before the operation or after the operation). The stack control bit sets up the index register to act like a stack. On read operations the index register is pre incremented and the value read from memory. On write operations data is written to memory and the index register is post decremented. The index count enable/disable bit is used to control if the associated count register is decremented as well. Note that the count register is always decremented by one.

Detailed Description Text (140):
The three bit circular buffer control bits are used to set up the index register as a circular buffer. A value of zero disables this function. The other seven states represent the size (n) of the circular buffer as a power of two (n+2). The size ranging from four bytes to five hundred twelve bytes. The bits are used as a mask when the index register is incremented/decremented. The buffer must be aligned according to its size.

Current US Cross Reference Classification (3):
711/110

CLAIMS:

3. An integrated circuit device according to claim 1 wherein each of said processors further has a load/store unit and further wherein each of said index control registers receives and retains a data bit directing whether the associated index register is to be incremented or is to be decremented on access to memory by the associated load/store unit.

6. An integrated circuit device according to claim 1 further comprising an plurality of index count registers each formed on said substrate and operatively associated with a corresponding one of said index registers and further wherein each of said processors further has a load/store unit and further wherein each of said index control registers receives and retains a data bit directing whether the associated index count register is to be automatically decremented in reponse to an access of the

associated index register.

10. An integrated circuit device according to claim 2 and further wherein each of said index control registers receives and retains a data bit directing whether the associated index register is to be incremented or is to be decremented on access to memory by the associated load/store unit.

13. An integrated circuit device according to claim 2 further comprising an plurality of index count registers each formed on said substrate and operatively associated with a corresponding one of said index registers and further wherein each of said index control registers receives and retains a data bit directing whether the associated index count register is to be automatically decremented in reponse to an access of the associated index register.

19. An integrated circuit device according to claim 10 further comprising an plurality of index count registers each formed on said substrate and operatively associated with a corresponding one of said index registers and further wherein each of said index control registers receives and retains a data bit directing whether the associated index count register is to be automatically decremented in reponse to an access of the associated index register.

24. An integrated circuit device according to claim 17 further comprising an plurality of index count registers each formed on said substrate and operatively associated with a corresponding one of said index registers and further wherein each of said index control registers receives and retains a data bit directing whether the associated index count register is to be automatically decremented in reponse to a n access of the associated index register.

28. An integrated circuit device according to claim 23 further comprising an plurality of index count registers each formed on said substrate and operatively associated with a corresponding one of said index registers and further wherein each of said index control registers receives and retains a data bit directing whether the associated index count register is to be automatically decremented in reponse to an access of the associated index register.

WEST

Generate Collection | Print

L9: Entry 21 of 30 File: USPT Apr 22, 1997

US-PAT-NO: 5623621
DOCUMENT-IDENTIFIER: US 5623621 A

TITLE: Apparatus for generating target addresses within a circular buffer including a register for storing position and size of the circular buffer

DATE-ISSUED: April 22, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Garde; Douglas	Dover	MA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Analog Devices, Inc.	Norwood	MA			02

APPL-NO: 08/ 115915 [PALM]
DATE FILED: September 1, 1993

PARENT-CASE:
This application is a continuation of application Ser. No. 07/368,365, filed Nov. 2, 1990, now abandoned.

INT-CL: [06] G06 F 12/02

US-CL-ISSUED: 395/421.1; 395/437, 395/421.07
US-CL-CURRENT: 711/220; 711/110, 711/217

FIELD-OF-SEARCH: 395/400, 395/425, 395/428, 395/437, 395/421.09, 395/421.07, 395/421.1, 364/2MSFile, 364/9MSFile

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3461433	August 1969	Emerson	340/172.5
☐	3813652	May 1974	Elmer et al.	340/172.5
☐	3931611	January 1976	Grant et al.	340/172.5
☐	3999052	December 1976	Gooding et al.	235/153
☐	4169289	September 1979	Shively et al.	395/250
☐	4187549	February 1980	Bond et al.	364/746
☐	4202035	May 1980	Lane	364/200
☐	4251860	February 1981	Mitchell et al.	364/200
☐	4408274	October 1983	Wheatley et al.	364/200
☐	4432054	February 1984	Okada et al.	364/200
☐	4453209	June 1984	Meltzer	364/200
☐	4453212	June 1984	Gaither et al.	364/200
☐	4485435	November 1984	Sibley	395/400
☐	4623997	November 1986	Tulpule	370/85
☐	4627017	December 1986	Blount et al.	364/900
☐	4722067	January 1988	Williams	364/746
☐	4724479	February 1988	Kloker et al.	364/746
☐	4800524	January 1989	Roesgen	364/900
☐	4809156	February 1989	Taber	395/400
☐	4819165	April 1989	Lenoski	395/400
☐	4833602	May 1989	Levy et al.	364/900
☐	4908748	March 1990	Pathak et al.	395/400
☐	4935867	June 1990	Wang et al.	364/200

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
WO79/00035	February 1979	WO	

OTHER PUBLICATIONS

Pages 6-5, 6-6, and 6-22 through 6-25 of TMS320C30 data book Aug. 1988.

ART-UNIT: 232

PRIMARY-EXAMINER: Chan; Eddie P.

ASSISTANT-EXAMINER: Nguyen; Hiep T.

ATTY-AGENT-FIRM: Wolf, Greenfield & Sacks, P. C.

ABSTRACT:

The invention comprises a hardware constructed address generator for a circular buffer which can be of any size and be in any position in memory. The address generator calculates both an absolute value and a wrapped value and selects one in accordance with whether the wrapped value falls within the boundaries of the buffer.

17 Claims, 5 Drawing figures

L9: Entry 21 of 30 File: USPT Apr 22, 1997

DOCUMENT-IDENTIFIER: US 5623621 A
TITLE: Apparatus for generating target addresses within a circular buffer including a register for storing position and size of the circular buffer

Brief Summary Text (4):
Digital information processors frequently employ digital memory buffers to temporarily store information en route to another device such as an input/output device or processor. A buffer may be constructed of dedicated hardware registers wired together or it may simply be a dedicated section of a larger memory. Such digital information buffers can take many forms. One such form is known as a circular buffer. In circular buffers, the addresses for accessing locations in the buffers typically are generated by modifying the contents of a pointer register which is external to the buffer area which points to an address location within the buffer. When that address is needed on the address bus, it is output from the pointer address and the pointer is incremented (or decremented) by a predetermined amount so as to be ready for the next instruction cycle which accesses the circular buffer. In circular buffers, means must be provided for "wrapping" the address around when the increment (or decrement) causes the address in the pointer register to fall without the bounds of the buffer. In other words, means must be provided for causing the address generator for the buffer to generate modulo addresses with the modulus being the length of the buffer.

Brief Summary Text (6):
One such hardware implemented system is disclosed in U.S. Pat. No. 4,800,524. The apparatus described in U.S. Pat. No. 4,800,524 comprises three registers external to the buffer, including (1) an L register which contains the length of the buffer, (2) an A register which contains the last address accessed in the buffer (this is the pointer register) and (3) an M buffer which contains an increment (or decrement) value to be added (or subtracted) from the A register. The apparatus also comprises two separate adder/subtractors, the first of which generates an absolute buffer address which is simply the contents of the A register added to the contents of the M register and a second adder/subtractor which generates a wrapped address by either adding (if M is positive) or subtracting (if M is negative) from the absolute address generated by the first adder the length of the buffer. Additional logic selects either the absolute address or the wrapped address responsive to the carry bits from the first and second adders. If the carry bits indicate that the absolute address generated is outside the boundaries of the buffer, the wrapped address is used and placed in the A register ready for the next access. Otherwise, the absolute address is selected and placed in the A register. The invention disclosed in the U.S. Pat. No. 4,800,524 patent is limited, however, in that in order for the system of examining the carry bits to work, the lower K bits of the buffer's base address (lowest address) must be zero, where K is the number of bits required to represent the length of the buffer and the length of the buffer must be a power of two. These limitations can be extremely inconvenient in certain applications.

Brief Summary Text (14):
3) a Modify register, M, which is loaded with the increment (or decrement) value, and

Detailed Description Text (2):
FIG. 1 illustrates a typical circular buffer incorporated as part of a larger memory. The base address of the buffer is the lowest numbered address in the buffer. In the example in FIG. 1 this is address 20, represented as B in FIG. 1. The highest address in the buffer is designated as the end address which in FIG. 1 is address 39 and is indicated as E. A pointer into the buffer typically comprises a register containing

the address of the next location in the buffer to be accessed. This register will hereinafter be termed the index register, I. Typically in a circular buffer, after each access, the index pointer is incremented or decremented a predetermined number of addresses so as to be prepared for the next access into the circular buffer. The number of address spaces which the index pointer is incremented or decremented will hereinafter be referred to as the modify amount and is represented in FIG. 1 as M. It is common for the modify amount to be a fixed number which never changes. However, there are applications in which the modifier amount, M, may be varied.

Detailed Description Text (3):
As stated above, in a circular buffer, means must be provided for wrapping the index pointer around when the increment amount would cause the index pointer to exceed the bounds of the buffer. For instance, in FIG. 1, the index pointer is shown as pointing at address 35. If the increment amount, M, is 3, then the pointer will be updated to point to address number 38. However, on the next increment of 3, the index pointer would normally point to address 41 which is beyond the bounds of the buffer. Accordingly, means must be provided for ensuring that when the increment of the index pointer causes it to exceed the end address of the buffer, it is wrapped around to the base of the buffer. For instance, on the next increment of the index pointer, the index pointer should point to address 21 rather than address 41.

Detailed Description Text (4):
FIG. 2 shows an address generator according to the present invention for generating addresses for a circular buffer such as the buffer shown in FIG. 1. FIG. 2 shows an embodiment of the invention in which the increment, M, is always positive or always negative, as is typical. However, other embodiments in which the modifier can be positive or negative are possible and are described later herein. The address generator of the preferred embodiment of the present invention comprises four registers, termed L, M, I and B. The L register 12 is initialized by loading it with the length of the circular buffer. The M register 19 is loaded with a increment (or decrement) value. The B register 14 is loaded with the base address of the circular buffer, i.e., the lowest numbered address if M is positive, or the highest numbered address if M is negative. The I register, which essentially comprises the pointer into the circular buffer, is automatically loaded with the base address when the base address is loaded into the B register. As shown in FIG. 2, OR-gate 16 assures that when a B register write enable instruction (B WREN 78) is issued, the I register is also write enabled such that both the B and the I registers receive the base address being placed on the data bus 18.

Detailed Description Text (5):
In most circular buffer applications, the buffer pointer, i.e., the I register, is incremented the same amount every time. Accordingly, a number can be permanently stored in the M register 19. In situations where the increment, M, may vary, a multiplexer 20 is provided. Under the control of a processor (not shown), to avoid unnecessary obfuscation, multiplexer 20 can select as the increment either the data placed on the data bus by the processor or the output of the M register 19.

Detailed Description Text (7):
As the contents of the I register are output onto the data bus, they are also fed into one input of an adder 22. The other input of the adder is coupled to the output of the multiplexer 20 which contains the selected increment, M0 The value I+M is output from the adder 22 and fed to one input of adder/subtractor 27. At its other input, adder/subtractor 27 accepts the output of the L register which contains the length of the buffer. In the case where M is always positive, the adder/subtractor is set up to subtract L from I+M. However, if M is negative, then the adder/subtractor is set up to add the value of L to I+M.

Detailed Description Text (9):
The purpose of the above-described operation of adder 22, adder/subtractor 27, comparator 30 and multiplexer 32 is explained as follows. The value of I+M output from adder 22 represents the new absolute value of the pointer (i.e., the old pointer value plus the increment M, regardless of whether it is within the bounds of the buffer). If the absolute value is within the range of the buffer, then no "wrapping around" is necessary and it can be placed directly into the I register. However, if it is beyond the range of the buffer, then the length of the buffer, L, must be subtracted from the

absolute value in order to "wrap around" the address in modulo style. Adder/subtractor 27 calculates I+M-L whether it will be needed or not. Obviously, if I+M is within the buffer range, then subtracting the length, L, of the buffer from the absolute address will cause the address to be less than the base address of the buffer, thus indicating that "wrapping" is unnecessary. However, if I+M is beyond the bounds of the buffer, then I+M-L will be greater than or equal to the base address, B, of the buffer. Accordingly, comparator 30 determines if I+M-L is greater than or equal to the base address, B. If so, then I+M must have been beyond the range of the buffer and the comparator causes multiplexer 32 to place in the I register the value of I+M-L, rather than the value of I+M.

Current US Cross Reference Classification (1):
711/110

WEST

Generate Collection | Print

L9: Entry 25 of 30 File: USPT Oct 31, 1995

US-PAT-NO: 5463749
DOCUMENT-IDENTIFIER: US 5463749 A
**** See image for Certificate of Correction ****

TITLE: Simplified cyclical buffer

DATE-ISSUED: October 31, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Wertheizer; Gideon	Petach-Tikva			IL
Be'ery; Yair	Petach-Tikva			IL
Ovadia; Bat-Sheva	Herzeliya			IL
Gross; Yael	Tel-Aviv			IL
Perets; Ronen	Ramat-Gan			IL
Milstein; Yakov	Natanya			IL

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
DSP Semiconductors Ltd	San Jose			IL	03
DSP Semiconductors USA, Inc.	San Jose	CA			02

APPL-NO: 08/ 003640 [PALM]
DATE FILED: January 13, 1993

INT-CL: [06] G06 F 12/00

US-CL-ISSUED: 395/437; 364/DIG.1, 364/DIG.2, 364/238.6, 364/238.8, 364/926.1, 364/926.4, 395/421.08
US-CL-CURRENT: 711/110; 711/218

FIELD-OF-SEARCH: 395/400, 395/200, 395/250, 395/425, 364/DIG.1MSFile, 364/DIG.2MSFile

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4202035	May 1980	Lane	395/400
☐	4800524	January 1989	Roesgen	395/400
☐	4809156	February 1989	Taber	395/400
☐	4833602	May 1989	Levy et al.	395/400
☐	4935867	June 1990	Wang et al.	395/400
☐	5134695	July 1992	Ikeda	395/400
☐	5247645	September 1993	Mirza et al.	395/425
☐	5276827	January 1994	Delaruelle et al.	395/400
☐	5282275	January 1994	Andre et al.	395/400

ART-UNIT: 235

PRIMARY-EXAMINER: Harrell; Robert B.

ATTY-AGENT-FIRM: Bloom; Leonard

ABSTRACT:

An improved cyclical buffer having an integer M number of memory locations in respect of which a number STEP of consecutive memory locations are required to be accessed in a single operation and having a predetermined START location defining an initial memory location to be accessed. M is constrained to be an integer multiple of STEP and the k least significant bits of START are zero where k is the minimal integer satisfying the relation 2.sup.k >M-.vertline.STEP.vertline.. The result is the same as the general MODULO algorithm employed in conventional cyclical buffers but without the cost of implementing the complete MODULO function. An apparatus for generating successive addresses involves an ADDER and a k-bit COMPARATOR coupled via a MULTIPLEXER to an address register such that the k-least significant bits of the ADDER or M-.vertline.STEP.vertline. or 0 is fed to the k-least significant bits of the address register depending on the output of the k-bit COMPARATOR.

4 Claims, 4 Drawing figures

L9: Entry 25 of 30 File: USPT Oct 31, 1995

DOCUMENT-IDENTIFIER: US 5463749 A
**** See image for Certificate of Correction ****
TITLE: Simplified cyclical buffer

Brief Summary Text (8):
The processing mechanism may be stepped whereby the cyclical buffer addressing mechanism is incremented or decremented by an amount equal to STEP after each access so that the elements are processed starting with the element location START, proceeding to START+STEP, and so on as far as END and then repeating the cycle in "STEP" increments as required.

Detailed Description Text (10):
If so, then the k least significant bits of the address arc set to zero, thereby causing the cyclical buffer to point to the address associated with the START location. In this connection, it will be recalled that the cyclical buffer 10 has M addressable locations. Thus, the current address location may be incremented by the value of STEP up to and including M-.vertline.STEP.vertline., whereupon incrementing the address further by the value of .vertline.STEP.vertline. would point to an address beyond the end of the cyclical buffer 10. The k least significant bits of the address are therefore set to zero corresponding to the START location.

Detailed Description Text (11):
If, on the other hand, the value of the k least significant bits of the current address is less than the value of M-STEP, then the current address is simply incremented by the value of STEP in order to point to the next memory location in the cyclical buffer 10.

Detailed Description Text (13):
If the value of the k least significant bits of the current address is not equal to zero, this means that we have not yet reached the end of the cycle mechanism (i.e. the start location of the cyclical buffer), and in this case the next address is simply determined by incrementing the current address by the value of STEP which, being negative, actually decrements the k least significant bits of the address.

Detailed Description Text (38):
Since, as before, each addressable location is determined by calculating only the k least significant bits thereof, the two most significant bits are unimportant so far as the calculation of successive addressable locations is concerned except insofar as they are, of course, calculated relative to the predefined START location which must be successively incremented by STEP until the end of the cyclical buffer is reached.

Current US Original Classification (1):
711/110

CLAIMS:

1. A cyclical buffer having an integer M number of memory locations, an integer STEP defining a number of consecutive memory locations, M being an integer multiple of STEP, and a predetermined START location defining an initial memory location defining a lower boundary of the cyclical buffer; and further including an addressing means for attaining improved buffer accessibility by setting, as many times as required, a current address to an address of successive memory locations whose contents are accessible, comprising:

least bit addressing means for addressing at least k least significant bits of the buffer where:

k is the minimal integer satisfying the relation 2.sup.k >M-.vertline.STEP.vertline.; and

the k least significant bits of START are zero; and the least bit addressing means includes:

incrementing means for incrementing the current address by STEP if STEP is positive and the k least significant bits of the current address are not equal to M-.vertline.STEP.vertline.,

said incrementing means further for incrementing the current address by STEP if STEP is negative and the k least significant bits of the current address are not equal to 0,

zero setting means for setting the address to zero if STEP is positive and the current address is equal to M-.vertline.STEP.vertline., and

decrementing means for setting the address to M-.vertline.STEP.vertline. if STEP is negative and the k least significant bits of the current address are equal to 0.

4. In a cyclical buffer having an integer M number of memory locations each having at least k addressable locations, an integer STEP defining a number of consecutive memory locations, M being an integer multiple of STEP, and a predetermined START location defining an initial memory location defining a lower boundary of the cyclical buffer wherein:

k is the minimal integer satisfying the relation 2.sup.k >M-.vertline.STEP.vertline.; and

the k least significant bits of START are zero; a method for attaining improved buffer acccssibility by setting as many times as required the current address to an address of successive memory locations, comprising the steps of:

if STEP is positive and the k least significant bits of the current address are not equal to M-.vertline.STEP.vertline., incrementing the current address by STEP;

if STEP is positive and the current address is equal to M-.vertline.STEP.vertline. setting the address to 0;

if STEP is negative and the k least significant bits of the current address are not equal to 0, incrementing the current address by STEP; and

if STEP is negative and the k least significant bits of the current address are equal to 0, setting the address to M-.vertline.STEP.vertline..